# Data Scraping

## Why it was needed

Data scraping was a necessary part of the team project as it was stated on the project brief that it should be used as we had no access to the data of the SHR website otherwise, we were not given access to the database in anyway. So we had to come up with a data scraping solution to solve this.

## What is Data Scraping

The process of data scraping is to obtain data from a website that was intended to be viewed by the end user, not for input to a program. The absolute simplest way to obtain data from a website would be to just do it by hand, go to the website and note down the data you want or copy and paste the needed information to a file. This would make sense if you only need a few lines of data, when a large data set is needed this would be a very bad choice to obtain that data.

Data scraping can also refer to using a html parser which would look through the html of the target website and select the data that is wanted through selecting the relevant tags. The data within the relevant tags would then be added to a useful data structure for use in whatever way it is needed for the particular circumstance.

Using an html parser or otherwise has its drawbacks however, in that if the design of the target website changes there is potential to render the whole scraping program useless. If the html parser/scraping program relies on a tag like:

<table id="tblGrid" cellspacing="0" cellpadding="0" align="Center" style="background-color:White;height:800px;width:790px;border-collapse:collapse;">

If the id of the tag was to change in any way from “tblGrid” then the whole table full of data that would be the target for scraping would no longer be accessible until maintenance on the scraping program had been made.

## How the data is presented

The required data on the SHR website for the most part is presented in tables. And the needed data within those tables is available within the <td> tags. Depending on what section of the SHR website is being accessed the table is paginated, to allow the data to be easier to read as opposed to having to scroll down through the entire data set.

The data available for the runners on the website is presented in table which is paginated to only show 75 runners at a time and the rest of the data is accessed through the clicking of links which are controlled by a JavaScript method “doPostBack()”.

The data for what races are available and the data associated with an individual runner on the site however is just shown in its entirety meaning there is no need to interact with the previously mentioned JavaScript method.

## The choices

In order to perform the scraping of data we had to look in to possible choices to achieve the desired result. We looked at a few different methods. The first method that was looked at was using PHP and to code each tag target in individually, which would take a lot of time and would increase the chances of human error in scraping, the only real reason this was considered was that one of the team members had used such a system previously.

However due to the amount of data needed and the way in which it was presented, PHP seemed unsuitable for the data scraping. The next choice was JSoup which is a Java library that has the sole purpose of extracting and manipulating data. After a relatively short time the basic scraping set up for a page of runners was achieved and this was why it was chosen to be used.

## Jsoup

Jsoup([jsoup.org](http://jsoup.org/)) is a Java library that was designed in order to allow its users to extract data from websites and manipulate that data. It has a few features that make the act of scraping data relatively simple. Its main feature is that it uses html parsing to select what data is required by providing the html tag that surrounds the necessary data.

It has the basic functions that would be expected of such a library such as the connect function to allow the actual connection to the website. Then there is the select method. The select method is basically a CSS selector that allows the user to specify exactly what piece of HTML/CSS they wish to extract.

The way the html parsing works in Jsoup is that, for instance, if you need to select all the links within in a page you would simply have “ links = doc.select("a[href]"); “. This would go through the html and find every tag that features the link in the format of “<a href=[hyperlink]>”  
If you wanted to select all the links that feature a specific type of value in the href part then you would have “links = doc.select("a[href\*=RaceDetails]");” this would select all of the links that lead to a webpage with “RaceDetails” involved at the start of the link.

## How the scraping program works.

When the scraping process is started the necessary folder structure is created to save the CSV files to. Then a cached thread pool is created in order to read through all of the runners data which is accessed by going to the individual runners page and scraping the data based on the specified html tags. The scraping of the runners data has been multithreaded as the access time to obtain all the data would take too long as it would require going to potentially several hundred webpages one after the other. With multithreading the access this means that several webpages can be loaded at once and scraped in unison. As some runners will have more data than others the process being multithreaded is beneficial in that the threadpool structure will allow for a thread to be assigned to another runner whilst the other thread is still working on obtaining the data from the larger data set.

Once each runner’s data has been obtained it is then stored within a CSV file named after the specific runner’s ID number and the runner’s ID is added to a hashset that is passed to a function that will generate all the races that the Westerlands club has ran. This is also written out to a CSV file. Each race access is also multithreaded using a cached thread pool to decrease running times in a similar fashion as the runner’s multithreading.

The individual race’s data is also scraped using the same method as before by selecting the relevant html tags.

## Thread Pools

When multithreading the access to the web pages a cached thread pool was chosen instead of a fixed thread pool. Both of these can be used to multithread processes.

A fixed thread pool allows you to specify exactly how many threads are to be made available for processing. If a fixed thread pool of 5 was to be used then 5 threads would be assigned a job each from a larger set of jobs, once each thread had finished its assigned job it would then take on the next available job and each thread would do this until all the jobs had been completed.

A cached thread pool works similarly in that threads are assigned jobs and new jobs are assigned to the thread on completion. The main difference, however, is that new threads are created as they are needed and when their job is completed the finished threads are assigned new jobs.

## The data obtained

The data that is currently being stored is data for a list of all the runners in the club, the individual runner’s information and the races that have been ran.

The list of all runners in the club stores the runner’s ID number, their name, average win percentage, races ran and their gender.

The individual runner’s data stores their position in a race, the name of the race, the date they ran the race, their finishing time and their percentage of winners time.

The races data stores the race ID, the name of the race, the venue of the race, the distance, the climb, the men’s record time, men’s year of the record, the men’s record holder’s name, the women’s record time, women’s year of the record, the women’s record holder’s name.

## Problems Faced.

One small problem that was faced in the beginning of the project was run time. At that stage we were focusing only on the first 75 or so runners. The scraping program took roughly thirty seconds to extract the relevant data for those runners. The time taken was attributed to the page load of the web pages more than anything else. While this didn’t seem a huge problem at the time, if it was left as it was it would lead to further problems later. Especially when dealing with a larger amount of data. The problem was solved by using a multi-threaded approach. Instead of visiting each runners page one after the other and dealing with the relevant load times of each we created a thread pool to give out that task as many times as was possible. A CachedThreadPool was used over a FixedThreadPool as it allows more threads to be created as needed and once one has completed its task it can be given the next task. This was chosen as it means the scraping program can run efficiently on different powered machines. A fixed thread pool could make the program run slower than it could potentially run on a more powerful machine. In implementing a thread pool the run has been significantly reduced.

The biggest problem that was faced and took some time to overcome was that on the SHR website the data that was being shown was paginated. The data was paginated in such a way that the scraping program couldn’t get all the necessary data because it was hidden behind a JavaScript function and the links that would change the page would involve data to be passed to the function. We needed to replicate this functionality to gain access to the data on the further pages of data. The data that was desired in the function was the event target and the event argument.

The problem with this function however is that Jsoup cannot simulate a mouse click of a user to allow access to the rest of the data and as none of us really had much experience with JavaScript it was quite a daunting task to deal with.

The JavaScript function that was doing the hiding is known as “\_\_doPostBack” and it is a function used in ASP.NET generated HTML. The function is as follows:

function \_\_doPostBack(eventTarget, eventArgument) {

if (!theForm.onsubmit || (theForm.onsubmit() != false)) {

theForm.\_\_EVENTTARGET.value = eventTarget;

theForm.\_\_EVENTARGUMENT.value = eventArgument;

theForm.submit();

}

}

The function takes in two arguments and they are the eventTarget and the eventArgument. The eventTarget is the id of the control that is submitting the form and the eventArgument is any additional information that may be needed to submit the form.

In the case of the paginated runners the eventTarget would be passing in the link that was clicked so that the page would be reloaded with the data assosciated with that tab. The eventArgument was blank in this case as no extra information was needed.

In order to solve the problem the Jsoup.connect() function needed to be passed in the link to connect to along with the data for the doPostBack() function, the eventTarget and eventArgument. It also needed to be passed the lastFocus, viewState, eventValidation and the userAgent.

The viewstate is basically a large block of encrypted text that stores within it the contents of the current page and the state of all the controls. The lastFocus in this case is blank. The eventValidation is there to provide a known set of possibilities of input to prevent the input of unknown/invalid data.

## Maintainability

As it stands the program is complete as there has been no need for other data at the moment. However if more data is required from the website it would be a reasonably simple process.